TWEAG
A Modus Create Company

TWEAG's Proposals for multiple core budget projects for Cardano 2025

EURL Tweag
207 rue de Bercy
75012 Paris, France

March the 31st 2025

Introduction and Overview

1. Tweag Information

Beneficiary Company
EURL Tweag, 207 Rue de Bercy, 75012 Paris, France, is a limited liability company.
Tweag is part of the Modus Create group of companies.
https://www.tweag.io/

Beneficiary
Benjamin Robin – General Manager
benjamin.robin@tweag.io
0033675989101

Submission Lead
Kris Kowalsky
Point of contact for Cardano budget
kristijan.kowalsky@moduscreate.com
00385913835058
https://x.com/KrisKowalsky
https://www.linkedin.com/in/kriskowalsky/

2. About Tweag and Modus Create

At Tweag by Modus Create, we offer a unique combination of deep technical expertise and strategic consulting capabilities, with a long-standing commitment to open-source and decentralized systems. EURL Tweag has been in business for over a decade, having started in 2013, and has built a reputation for engineering excellence across critical infrastructure projects.

Since January 2018, we have been continuously engaged with IOG (Input Output Global) on a variety of initiatives within the Cardano ecosystem, including conducting audits and core protocol development. Since May 2021, we have also provided formal audits for the Cardano ecosystem, helping to ensure the reliability and security of mission-critical code.

Our team has played a leading role in the development of Cardano's core infrastructure, including leading the consensus and ledger teams, implementing the Ouroboros Genesis protocol, and contributing to the design of Ouroboros Peras. We have been involved in nearly all aspects of the core Cardano node, giving us a comprehensive understanding of the system's

architecture, roadmap, and strategic direction.

Beyond Cardano, we bring deep, practical experience with Haskell, Rust, and a wide range of technologies used in polyglot projects, enabling us to engineer scalable, secure, and high-performance systems across domains.

As part of Modus Create, a global digital transformation consulting firm, we are backed by a diverse team of strategists, designers, and technologists who help the world's leading brands build digital advantage. Modus Create specializes in strategic consulting, full lifecycle product development, platform modernization, and digital operations and is an official partner of top-tier technology providers such as Atlassian, AWS, and GitHub. This global reach and cross-disciplinary strength provide our clients with unmatched capabilities throughout the full product development lifecycle.

This proposal reflects our ongoing commitment to advancing Cardano's mission through rigorous engineering, strategic alignment, and high-impact delivery.

3. Introduction

This document outlines Tweag by Modus Create's proposals for a suite of core infrastructure projects aimed at advancing the technical foundations of the Cardano blockchain. These initiatives focus on novel, high-impact developments across key areas of the Cardano stack. In the months ahead, Tweag intends to collaborate closely with qualified suppliers, DReps, and Intersect to further shape the structure and scope of these engagements.

The proposals presented here represent Tweag's intended contribution to Cardano's continued evolution. We invite other ecosystem participants to submit complementary or competitive proposals, and we are open to exploring collaborative partnerships where aligned.

We have consolidated these projects into a single proposal to provide a cohesive view of our core roadmap delivery. We plan to actively seek community input to evaluate whether any of the initiatives should be pursued independently and will adapt our structure accordingly based on that feedback.

Resourcing

The successful execution of these workstreams depends on a collaborative effort grounded in the deep technical expertise of Tweag by Modus Create and its longstanding network of trusted collaborators. Together, these teams bring a wealth of experience that spans the full history of

Cardano's development—from early protocol design and formal verification to high-assurance software engineering and real-world system deployment.

The broader effort will draw on the capabilities of a diverse and skilled group of contributors, including but not limited to: BCryptic, Blinklabs, Galois, Globant, IO Engineering, Mlabs, Obsidian Systems, Palo IT, PNSOL, Quviq, Serokell, SundaeLabs, TXPipe, Vacuumlabs, Well-Typed, and others. These collaborators represent some of the most experienced teams in the Cardano ecosystem, each with a proven track record in delivering mission-critical components.

Costing and Payments

The cost estimates presented in this proposal reflect our current best assessment for each initiative, based on their anticipated duration and the projected Full-Time Equivalents (FTEs) required from both Tweag by Modus Create and collaborating subcontractors. These estimates serve as a planning baseline, informed by our experience in delivering complex infrastructure projects within the Cardano ecosystem.

We recognize that project development often involves evolving requirements and technical challenges. In the event that a workstream's scope or complexity exceeds initial expectations, the designated Tweag project lead will escalate the matter to the Technical Steering Committee (TSC). This ensures that any proposed adjustments are subject to appropriate governance and due diligence before additional resources are allocated or budgetary changes are approved.

Tweag intends to receive compensation for delivered services in USD ($). The specific USD-to-ADA conversion rate will be finalized during the contracting phase. Tweag will also coordinate with Intersect to define and execute payment mechanisms for any subcontractors in accordance with the individual terms outlined in each agreement.

Administration and Contracting

Tweag by Modus Create requests that Intersect serve as the designated proposal and contract administrator following the roles and responsibilities outlined in the Cardano Constitution. This includes acting as the auditor of record, responsible for assessing progress and verifying that deliverables align with the commitments defined in this proposal.

Tweag anticipates entering into direct contractual agreements with Intersect, with a preference for Milestone Based Fixed Price contract structures where deliverables are clearly defined and scope is limited. Contract types, payment terms, and rate structures with subcontractors will be defined on a case-by-case basis.

Tweag retains full discretion over internal staffing and subcontractor selection, and reserves the right to onboard new or alternative suppliers as needed to ensure successful delivery. While specific security auditors are not identified in this proposal, subcontractors will be responsible for selecting, engaging, and funding security audits required to meet delivery standards.

Multi-Supplier Collaboration

Initiatives involving multiple suppliers will be delivered through close coordination between participating teams, with active engagement and guidance from the Technical Steering Committee (TSC). The TSC will support solution scoping, provide strategic oversight, and help ensure alignment with broader ecosystem priorities throughout the lifecycle of each workstream.

The inclusion of suppliers in this proposal reflects potential collaboration opportunities; however, it does not guarantee that any specific supplier will be engaged for delivery. Contracted suppliers will retain full autonomy over their internal resource allocation, work prioritization, and day-to-day project management responsibilities.

The Software Readiness Levels (SRLs) and release cadence for these initiatives will depend on the outcomes of funding decisions. Tweag anticipates working closely with Cardano development teams to align on release planning, technical milestones, and delivery expectations as proposals are executed.

Proposals

Context

This section holds project information as it is listed in the Intersect online form, here aggregated for easier readability.

No.	Project name
1	Peras
2	Canonical Ledger State
3	Black box Ledger Conformance Testing
4	Conformance Testing of Consensus
5	Plutus Script Re-Executor
6	Genesis Sync Accelerator
7	Canonical Block and Transaction Diffusion Codecs
8	Hoarding Node
9	Block Cost Investigation
10	Cardano-node-emulator proposal
11	History Expiry

Project name: Peras (partial)

Section 3: Problem Statements and Proposal Benefits

Problem Statement

Ouroboros Peras (in its pre-alpha version) improves settlement guarantees under optimistic conditions. We refer to CPS-0017 for more context on settlement.

Proposal Benefit

Ouroboros Peras (in its pre-alpha version) improves guaranteed settlement from hours to minutes under optimistic conditions (high honest participation and low adversarial activity) compared to the status quo. The exact guarantees depend on the chosen parameterization and the resulting trade-offs.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Yes, Finality (Peras) is listed in the roadmap.

Does your proposal align to any of the categories listed below?*

It supports the product roadmap

Committee Alignment

TSC

Supplementary Endorsement

TSC, Community

Section 4: Proposal Details

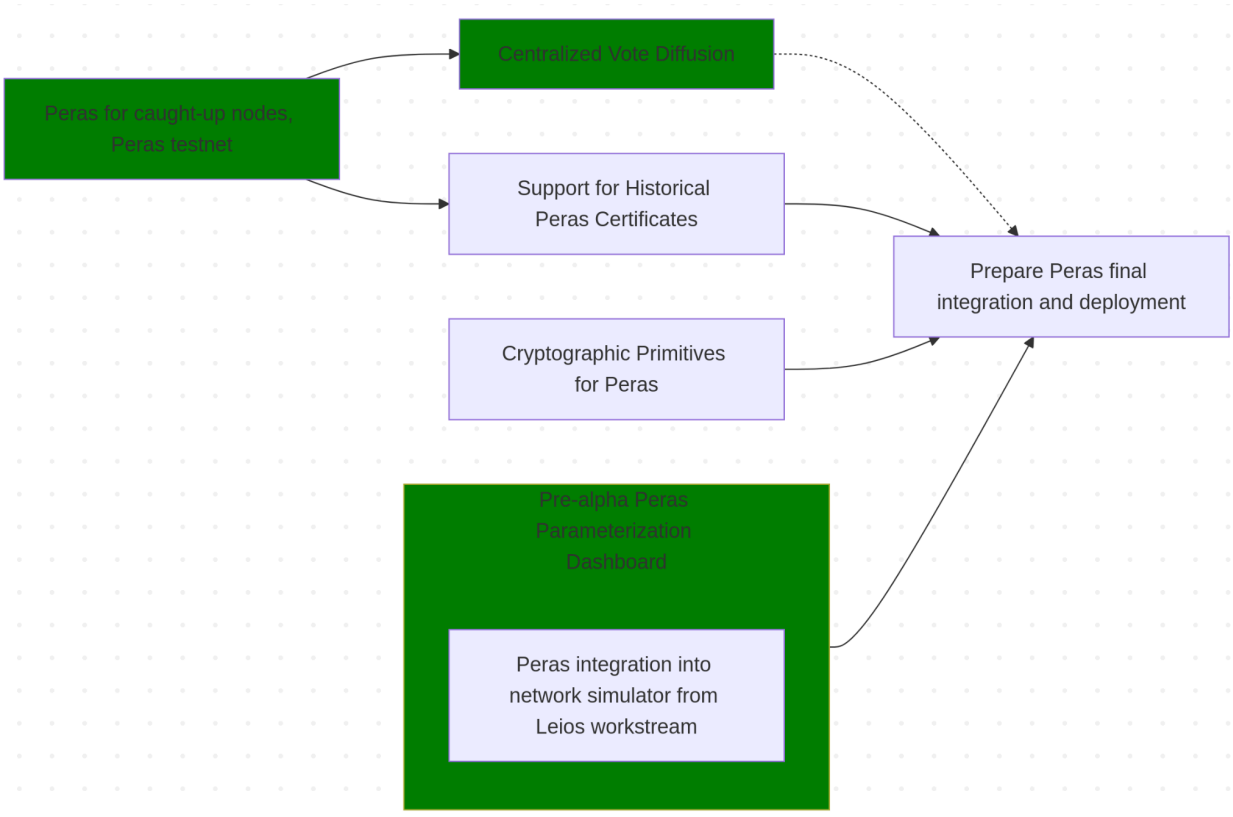
Proposal Name

Peras (partial)

Proposal Description

We refer to our report on the Peras design and architecture for the details.

Concretely, we propose to work on the following three tasks (colored in green) for Peras (but a different subset, both smaller and larger, is conceivable, with potential for parallelization). Their relation and dependencies to each other and the remaining work is depicted below. There are the three necessary to deploy and experiment with the first Peras testnet, for example.



Pre-alpha Peras Parameterization Dashboard

Pre-alpha Peras features subtle tradeoffs in its parameter space, most prominently:

- Peras round length (and hence speed of settlement) vs additional network traffic.
- Peras block selection offset (and hence settlement latency) vs resistance against weak adversaries causing long cooldown periods during which Peras is ineffective.
- Length of cooldown periods vs Praos safety.

Currently, an easy overview of the resulting interplay of parameters is not available. We propose to rectify this situation via an easy-to-use web-based dashboard[1], substantiated by further analysis of the network impact of Peras, such as integrating (a simple version of) Peras into the network simulators developed as part of the Leios workstream.

Peras for caught-up nodes, Peras testnet

We propose to implement the aspects necessary to run Peras amongst caught-up nodes (see "Prototype without historical certificates" in the Peras design report). This allows to set up a dedicated testnet for Peras, enabling stake pool operators as well as potential users of Peras to assess its dynamics and guarantees in a realistic setting.

This prototype will likely use stub/unaudited cryptography for Peras votes and certificates.

Centralized Vote Diffusion

As an intermediate approach to reduce the network traffic impact of Peras, it is possible to let the Peras vote traffic be handled by a centralized party[2]. Concretely, pools only have to diffuse their own vote (if they are part of a Peras committee) to a centralized party, which requires only trivial bandwidth. This centralized party would then inject the resulting certificate back into the decentralized network, which would diffuse it internally.

Clearly, this centralized party has the power to disable Peras and therefore its improved settlement guarantees (by not diffusing any certificates); however, they can not threaten the safety/liveness of Cardano as a whole. We envision this centralized aspect to be temporary only, easing the adoption of Peras on the Cardano mainnet.

For a later decentralized version of vote diffusion, using a P2P network separate from that for block diffusion has certain advantages (such as being optimized for many-to-many diffusion with throughput instead of latency constraints); so the centralized vote diffusion approach is already similar in that regard.

[1] The existing Peras dashboard only gives a relatively narrow picture. Another point of reference is the Leios cost estimator.
[2] Similar in principle to how Mithril initially (and still until today) has a centralized aggregator, albeit for somewhat different reasons.

Dependencies

Discussions with IO Peras researchers
Input from potential Peras users and SPOs

Maintenance

Minor; the Peras implementation will likely live in a branch at this point, so the main maintenance work consists of regular rebasing.

Key Proposal Deliverables

- An intuitive dashboard to help key stakeholders (potential users of Peras, stake pool operators) make an informed decision whether pre-alpha Peras has an acceptable cost-benefit ratio.
- A prototype implementation of the aspects of Peras affecting caught-up nodes, allowing for a Peras testnet.
- Design and prototype of a simple mechanism for centralized vote diffusion, allowing to minimize the network traffic impact of Peras as part of a phased rollout.

Resourcing & Duration Estimates

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Peras	7	10	$27,680	$1,937,600

Project name: Canonical Ledger State

Section 3: Problem Statements and Proposal Benefits

Problem Statement

The current Cardano ledger state is defined internally to the cardano-ledger Haskell project. While the ledger specification sets out precisely how the state is updated in response to block and transaction payloads, the serialised format is only specified by its implementation and acts only as a dump of the in-memory representation. The ledger team is free to update this format to meet the needs of the ledger - for example, altering the representation of certain types in order to make lookups faster.

This has historically been an issue for tools such as db-sync, which have relied on exploring dumps of the ledger state to populate its database. It additionally complicates the situation for tools such as Mithril, which would like to distribute signed snapshots of the ledger state.

The problem, therefore, is that there does not exist a canonical representation of the ledger state suitable for:

- Signing and distribution
- Treating as a stable format to build additional tools against

This proposal would define such a thing.

Proposal Benefit

Having a canonical ledger state serialisation would bring the following benefits:

1. Any tool needing to extract information from the ledger state could now be written against a stable, documented and well-defined format.
2. Tools such as Mithril could explicitly sign and publish snapshots of the ledger state.
3. Multiple node implementations could explicitly share versions of their ledger state. This would be extremely useful for conformance testing between nodes, and setting up integration tests for validating implementation correctness.
4. Since our canonical format would support explicit versioning, it would be much easier to directly upgrade the ledger state between versions of the existing Cardano node. Currently, this process often requires replaying the chain in order to match ledger state changes.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Multiple Node Implementations

Does your proposal align to any of the categories listed below?*

Core

Committee Alignment

TSC

Section 4: Proposal Details

Proposal Name

Canonical Ledger State

Proposal Description

We would define a canonical ledger state serialisation format for Cardano that would exist externally to any given node implementation. Such a format would be defined in such a way as to:

- Be easily usable from multiple languages and platforms.
- Support efficient conversion to/from the existing ledger state in cardano node, and, to the extent already defined, those of Amaru, Dingo etc.
- Support explicit versioning of ledger states.
- Allow the provision of extended information which may not be needed by all nodes.
- Support efficient and decomposable hashing. It will be possible to independently hash certain parts of the ledger state known to require it, such as the stake distribution or the protocol parameters. Careful consideration will be given to ensuring that the hash is computable in a streaming fashion for large structures which may reside on disk already.
- Play nicely with the existing work on UTxO-HD/LSM trees. It will be possible to easily build the canonical representation (or simply its hash) from the on-disk LSM representation and vice versa.
- Support additional, non-hashed metadata, such as which program (including version) created this snapshot.

Dependencies

No explicit dependencies, but we would want to interact with the following:

- The UTxO-HD stream, in order to be compatible with the LSM work
- Cardano-ledger in general

- The greater extent we can collaborate with other node implementations, the better the result is likely to be for them.

Maintenance

After delivery, the canonical representation will need to be collectively owned by the various node implementations. It will need to be updated along with any changes to the ledger.

Key Proposal Deliverables

1. An appropriate specification for a serialised ledger state. Most likely this would be a CDDL definition, but we may also explore other serialised formats if there is a strong argument for not being consistent with other serialised Cardano data structures.
2. Conversion code suitable for converting to/from the existing Cardano node's ledger format.
3. Additional tooling to work with the serialised format, providing:
    a. Access to metadata (e.g. what block point is this state associated with, what node produced it).
    b. Ability to verify the snapshot against internally (one may embed the hashes in the metadata) or externally provided hashes or signatures.
    c. Access to subcomponents of the ledger state or their hashes.
    d. (Optionally) Support for directly querying by UTxO, address etc.
4. Tests for all of the above.

Resourcing & Duration Estimates

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Canonical Ledger State	3	4	$27,680	$332,160

Project Name: Black box Ledger Conformance Testing

Section 3: Problem Statements and Proposal Benefits

Problem Statement

The Cardano ledger formal specification represents the gold standard in terms of defining and testing an implementation. From the formal specification can be generated code which allows full conformance testing covering all potential scenarios and features.

However, making use of this requires considerable investment - one must extract the language-specific bindings from the Agda specification, define appropriate translations to/from the executable specification, and build appropriate transaction generators to exhibit all the features one might wish to test.

Other chains get away with something simpler; ethereum for example has a suite of tests (https://github.com/ethereum/tests) for a proposed client implementation to test against. These are not as rigorous as the formal specification but nonetheless provide a decent degree of assurance the multiple node implementations are executing the same ruleset.

With the provision of a canonical ledger state for Cardano (separate proposal) it becomes feasible to define a similar suite of tests covering the implementation of the Cardano ledger.

Proposal Benefit

With the provision of a black-box testing suite for the ledger state transition, doing simple verification of a new node becomes as simple as:

1. Define conversions to/from the canonical ledger state.
2. Run your ledger transition function given a snapshot and a sequence of transactions (plus some metadata like block/slot numbers).
3. Compare the result against the recorded ledger snapshot.

Further, we would provide tools to generate new test cases from existing test frameworks within the cardano ledger. This would allow new node implementations to benefit from the conformance testing against the formal specification without themselves needing to interact with the Agda tooling.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Multiple Node Implementations

Does your proposal align to any of the categories listed below?*

Core

Committee Alignment

TSC

Section 4: Proposal Details

Proposal Name

Black box ledger testing

Proposal Description

We would build on the existence of a canonical ledger state serialisation format to build a suite of tests covering the main ledger transition function.

This would take the form of a repository containing test cases, where each test case consists of:

- An initial state snapshot
- A sequence of transactions or blocks
- A final state snapshot

In addition, we would provide a tool to perform intelligent diffing of the state snapshot and provide guidance as to which parts have changed and how.

We would also contribute tools to cardano-ledger to support generating black box test cases from the existing suite of ledger tests. These would allow the black box testing to be easily updated should new features be added to the ledger.

Dependencies

- Canonical ledger state
- Some access to the cardano-ledger team would be very helpful in extracting test cases from existing tests.

Maintenance

Tests will be appropriately versioned (to the major protocol version), meaning that they should not go stale. Additional tests, and for new protocol versions, could be contributed easily through the use of the tooling we will build for cardano-ledger.

Key Proposal Deliverables

1. A library of test cases covering all the major features of the Conway era. (We do not propose covering prior eras, since most node implementations are not intending to implement prior era logic).
2. An intelligent diffing tool for the canonical ledger state.
3. Tooling in cardano-ledger to support automatically generating new test cases from ledger conformance tests.

Resourcing & Duration Estimates

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Black box Ledger Conformance Testing	3	4	$27,680	$332,160

Project name: Conformance Testing of Consensus

Section 3: Problem Statements and Proposal Benefits

Problem Statement

The eponymous feature of a consensus protocol is, of course, that it is expected to maintain consensus. In our terms, all nodes (subject to the various assumptions on the state of the network and honest majority) will agree on a prefix of the current chain.

Up to now, this has been achieved in the following ways:

- Every node in the network is running (close to) the same code.
- Consensus testing (living inside io-sim, a Haskell IO simulator) validates that all (honest) nodes will eventually reach consensus.

In a world of multiple node implementations, this strategy no longer holds. Nodes may be running very different code, and most of it will not be testable under IO-sim. Nodes failing to agree on the correct chain risks an accidental hard fork. Should one persist long enough it might potentially be unrecoverable.

We need to revisit consensus testing in the context of alternative node implementations.

Proposal Benefit

This proposal will extract portions of the Consensus tests of the existing node that will be of chief benefit to two groups:

- Implementors of alternate nodes, giving them a means of validating that they have implemented the consensus protocol stack correctly.
- The wider Cardano community, in helping to ensure that the network does not end up with an accidental hard fork.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Multiple Node Implementations

Does your proposal align to any of the categories listed below?*

Core

Committee Alignment

TSC

Section 4: Proposal Details

Proposal Name

Consensus Conformance Testing

Proposal Description

During the course of implementing Ouroboros Genesis, we designed an approach to node testing which we now call “Node vs Environment”. In effect, while we are ultimately interested in the behaviour of multiple nodes agreeing on the “right” chain, we can more easily test by taking advantage of two insights:

1. The logic of identifying the honest chain *locally* is tricky, but it is very easy to identify *globally*. Since we are only interested in cases where there is a global best chain, we have a very simple judgment rule as to whether a node has selected the correct one.
2. Once we have an easily identified honest chain, we no longer need to simulate multiple nodes and look for agreement - instead, we simulate a single node and judge the correctness of its responses to stimuli.

The result of this approach for Ouroboros Genesis was a testing framework that makes use of a single coordinated *point schedule* in order to simulate multiple upstream peers (possibly adversarial, possibly colluding) and validate that a syncing node ends up with the correct chain.

Whilst the point schedule currently is implemented inside the Haskell node, its declarative nature makes it possible to export this testing method and make it usable across diverse node implementations. To this end, this proposal would aim at the following:

1. Refine the point schedule generators to focus less on syncing nodes (which was originally appropriate for Ouroboros Genesis).
2. Define a serialised format for consensus test cases, covering the chain and point schedule.
3. Extract from the current node testing suite a standalone capability to generate and export point schedules, along with their underlying chains.
4. Create an independent utility that does the following:
    a. Read a serialized point schedule.

b. Act as one or more peers serving points as defined on the schedule. Such peers would instantiate appropriate protocols to serve as upstream peers to the node under test.
c. Open a timing socket to allow the node under test to control the “ticking” of the schedule.
d. Shrink and restart test cases upon signal from the node under test.
e. Re-export shrunk test cases to disk — failing test cases that have been shrunk are much easier for developers to debug.

This infrastructure would allow other node implementations to take advantage of the work already done to test the Haskell node, as well as allowing them to validate compatible consensus behaviour.[3]

Dependencies

- The Consensus Team will need to review the Pull Requests that enrich the point schedule generator (and also add corresponding tests to the existing test suite).
- The architects of alternative nodes would provide very useful feedback at a few phases throughout the work.

Maintenance

- As the Consensus algorithm expands—Peras, Leios, etc—it would make sense to enrich this testing strategy. However, even if that did not happen immediately, maintaining tests with just a focus on Praos remains important, since Praos is still the foundation that Peras and Leios enrich.
- It is useful to note that the exact interface of the nodes under test must be relatively stable, since it’s primarily the same interface that they use to communicate with peers on mainnet.

Key Proposal Deliverables

- Point schedule generators that are less focused on the syncing node.
- A design for how to manage time in this derived testing setup that no longer benefits from the io-sim framework.
- The utility for simulating a node’s upstream peers according to the generated point schedule and also shrinking that point schedule upon failures.

Resourcing & Duration Estimates

[3] Another synergy consists with the Tartarus project https://github.com/cardano-scaling/tartarus

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Conformance Testing of Consensus	4	9	$27,680	$996,480

Project name: Plutus Script Re-Executor

Section 3: Problem Statements and Proposal Benefits

Problem Statement

Developers of a decentralized application (dapp) would ideally have the option to easily analyze the successful executions of their scripts on the Cardano chain, if only to keep some off-chain state synchronized. To reduce load on the network, the fee formula incentivizes script authors to minimize the script's logic and the size of the state it maintains. Unfortunately, this means state that does not influence the script's validity judgment but is otherwise useful to the dapp developer (eg an individual user's history of usage, statistics about general usage, etc) is not automatically available on-chain. Dapp developers must use auxiliary tooling in order to maintain that state off-chain such as the one proposed here.

This proposal would deliver a monitoring tool for that particular purpose that does not need to maintain its own full copy of the ledger state and so is significantly easier to deploy. Any tool that would otherwise need to maintain its own copy of the ledger state would benefit from the work described herein, but this initial proposal maintains a narrow focus on being able to re-execute a Plutus script with the same settings and arguments (aka the correct ScriptContext) as it was executed on-chain.

The Extended UTxO Model is a key foundation of the Plutus platform, but also fundamentally poses a specific challenge to the task of re-executing scripts: every UTxO that a transaction lists within its inputs is passed to every script required for that transaction's validity. In particular, each script receives Plutus-representations of all the UTxOs lists as the transaction's inputs, regardless of the script's particular purpose (eg spending one such UTxO). As a consequence, dapp developers cannot anticipate whether some fresh UTxO will be relevant to future executions of their script—unless their script forbids the transaction from spending any UTxO beyond some ad-hoc predictable set, but that restriction would likely make their script burdensome and unpopular.

For this reason, access to the entire UTxO set is required in general to re-execute a script within the same settings and arguments as the Cardano node would execute the script. The node's existing GetUTxOByTxIn local query suffices for accessing this data, but the node does not retain UTxO sets older than 2161 blocks—which usually arise in ~12 hours—so scripts could not necessarily be re-executed more than 12 hours later. For example, to execute a script as it was executed inside a block one week ago would require the UTxO set from one week ago, which a node could only provide access to if it were one week behind the network.

The only general option for dapp developers today is to maintain a second ledger state, using the local node as the source of blocks. However, running that separate process incurs costs of its own.

- The ledger state requires a significant amount of RAM. Satisfying both the node's memory requirements as well as that of the monitor process may increase the developer's infrastructure costs. This burden is significantly reduced by UTxO HD, but doubling the memory cost of the ledger state is likely to always be undesirable.
- With UTxO HD, the monitor process will similarly require gigabytes of additional disk space. Today the UTxO HD footprint is dwarfed by the node storing all of the blocks, but in the future the UTxO HD footprint itself might grow to a significant enough size that a second copy is undesirable.
- Etc.

Proposal Benefit

This proposal would enable a more convenient experience for the Cardano dapp developer that needs to monitor the on chain execution of their scripts. In particular, this would enable off-chain analysis of on-chain script execution. The infrastructure would automatically maintain synchrony between the node and the monitor while minimizing redundant resource utilization.

In particular, the re-executor need not actually execute the same script. A major benefit is that it can instead execute an enrichment of the script that yields additional tracing information about the actual script's internal dynamics. Such data does not truly belong on-chain and would increase fees if it were, but it can be very useful to dapp developers.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Developer/User Experience

Does your proposal align to any of the categories listed below?*

None of These

Committee Alignment

TSC

Section 4: Proposal Details

Proposal Name

Plutus Script Re-Executor

Proposal Description

Dapp developers would use the proposed tool as follows.

- The dapp developer would run both a node following the Cardano network's chain as well as the re-executor tool.
- The re-executor's configuration data would include a map from script hash to the corresponding Plutus Core.
    - HOWEVER, this supplied Plutus Core can differ from the Plutus Core executed on-chain! In particular, it could be the result of compiling the same Plinth source code without providing the –remove-trace flag. In this way, the dapp developer could observe arbitrary tracing indicating their script's actual on-chain execution dynamics without any Cardano node needing to waste computational costs on those traces or inflate the transaction's fees accordingly.
- The dapp developer's arbitrary downstream tooling will process the re-executor's events.
    - The re-executor will emit an event each time a script identified in the configuration map is executed on-chain, including details about the corresponding re-execution. The event would also include the slot number, block number, and header hash.
    - The re-executor will also emit an event each time the local node updates its selection, including the slot number, block number, and header hash, which is sufficient information for the dapp developer to estimate of the on-going settlement probability for each script execution, according to standard Cardano settlement tables. (Kupo has some short and nice documentation about "Rollbacks & chain forks", for context.)
    - The re-executor will additionally emit events identifying which script executions were undone whenever the local node switches away from the blocks that executed some scripts in the configuration data.

In order to provide those events to the dapp developers, the re-executor and local node will communicate as follows.

- The local node will push valid blocks to the re-executor via the Node-to-Client ChainSync mini protocol as it selects those blocks. Whenever the re-executor processes a block including IsValid=True transactions that executed Plutus scripts with hashes given in its configuration data, it would fetch the necessary data from the ledger state via queries such as GetUTxOByTxIn, execute the corresponding given Plutus Core with the same exact arguments (aka by rebuilding the correct ScriptContext) as the on-chain script execution, and emit an event that identifies each re-execution and includes its observed trace messages.
- In order to identify which transactions execute a script that matches one of the given hashes, the re-executor will either need to maintain some auxiliary indexes as it processes blocks to know which transaction inputs are guarded by a relevant script or— eg if those indexes grow too large—the re-executor can fallback on/be configured to instead use significant additional GetUTxOByTxIn queries instead.

- The local node will not discard a ledger state until the re-executor has signaled it no longer needs to query that ledger state.
    - Such coupling requires alterations to the node.
    - The MsgAcquire command within the LocalStateQuery mini protocol already provides this in some sense. However, if the re-executor starts falling further and further behind, this existing mechanism would simply increase the node's memory usage without bound (disk space as well for UTxO HD).
    - Moreover, if the node restarts when the re-executor is retaining a historical ledger state, the re-executor would not be able to re-acquire that ledger state.
    - Designing the exact mechanism and ensuring it has a well-understood bounded detriment to the Cardano node and broader network as well as a tolerable experience for the dapp developer deploying it would be part of the first phase of the proposed work. One general extreme in this design space could rely on Ouroboros Genesis to follow the network's chain during intervals where the re-executor (or whatever arbitrary tool is locally connected via this mechanism) was slower than the Cardano chain's growth.

There are three key benefits to implementing this with the proposed architecture.

- It very clearly bounds how the node itself might be affected, which is important since that node is part of the Cardano network and moreover shares its source code with the rest of the network.
- It directly allows for a single node to serve multiple re-executors, even remotely if they have permission to tunnel in eg. To be clear: the slowest re-executor would determine whether that node can keep up with the Cardano chain's growth.
- It does not assume that the node and the re-executor must always both be online, well-connected, healthy, and in sync — it is unrealistic to impose such a 100% up-time requirement on dapp developers. Instead, the architecture ensures that the local node will pause while the re-executor is unavailable.

Dependencies

- It's plausible that much of the re-executor's logic could be reused from the cardano-ledger libraries if the Ledger Team is willing to include those parts within their stable interface.
- Similarly small requests might be submitted to the Plutus Team to ensure some functions useful to the re-executor are part of those libraries' stable interface.
- At least the Consensus Team will need to review the proposed alternation to the node.

Maintenance

- The re-executor would need an update in advance of hard forks, since each alters the codec for blocks and may alter the execution context for scripts.
- If the re-executor becomes popular, it would be worthwhile to parameterize how it emits script execution events: various EDA platform's formats, etc.

- If this general architecture for monitoring becomes popular for other use cases beyond the re-executor, requests for new LocalStateTxQuery queries might become more frequent.

Key Proposal Deliverables

1. A design for the re-executor and its necessary interface to the node.
2. A specification of the necessary node changes.
3. An argument bounding how those changes might increase the node's resource utilization or otherwise interfere with its primary responsibilities.
4. An implementation of those changes in the node.
5. An implementation of the re-executor itself.
6. A robust test suite.
7. Potentially also: requests to the Ledger Team and/or Plutus Team to tweak their libraries' stable interface.

Resourcing & Duration Estimates

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Plutus Script Re-Executor	4	9	$27,680	$996,480

Project Name: Genesis Sync Accelerator

Section 3: Problem Statements and Proposal Benefits

Problem Statement

Ouroboros Genesis (first version released in cardano-node 10.2) allows to sync the historical Cardano chain without having to trust any particular node. Concretely, this is achieved by connecting to a sufficiently large (~30) number of stake pool relays sampled from a recent stake distribution during catch up. As long as at least one of those peers is honest, the node will successfully complete its sync on the correct chain.

This process has already been optimized to only incur minimal load on all but one of those peers: fundamentally, the node needs to download all historical headers and blocks, and currently it does so by fetching them from one of the sampled stake pool relays.

- This extra load on stake pool relays may become excessive if many nodes sync via Genesis.
- Syncing nodes in geographical locations that do not feature many stake pool relays[4] might not find an available nearby stake pool relay, therefore suffering from subpar network performance.

We propose a simple tool that allows syncing nodes to download blocks and headers from centralized Content Deliver Networks (CDN) without reducing the trust assumptions of Ouroboros Genesis. If the CDN serves an incorrect historical chain, the syncing node will safely fall back to downloading the chain from the decentralized Cardano network.

Proposal Benefit

Reduced load on stake pool relays, as well as faster syncing via Genesis for nodes in remote locations.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Scaling the L1 Engine

Does your proposal align to any of the categories listed below?*

Core

[4] Currently, stake pool relays are greatly overrepresented in Europe and North America.

Committee Alignment

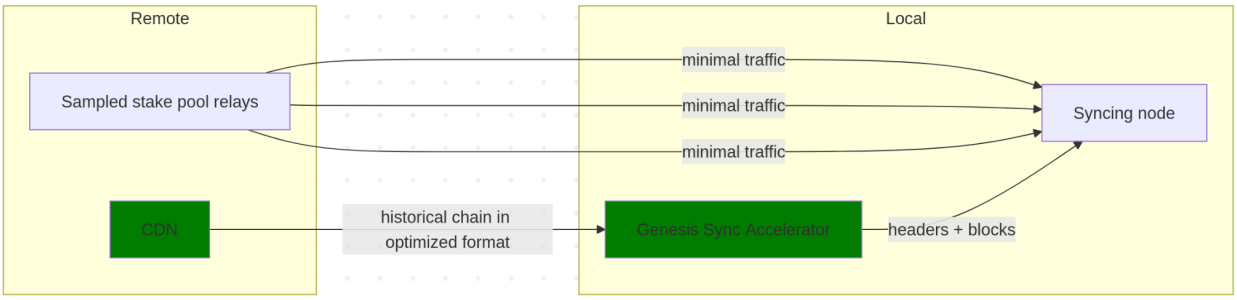
TSC

Section 4: Proposal Details

Proposal Name

Genesis Sync Accelerator

Proposal Description



The Genesis Sync Accelerator is an adaptor (speaking the inbound Node-to-Node protocols) that a syncing node (via Ouroboros Genesis) fetches all historical headers and blocks from, as long as the minimal additional traffic from the sampled stake pool relays confirms that this is indeed the correct historical chain. This will require minor changes to the existing logic for choosing which peer to download headers and blocks from (such that the Genesis Sync Accelerator can be prioritized when appropriate).

In turn, the Genesis Sync Accelerator obtains a historical chain from a *potentially untrusted* source. A natural candidate is a (potentially compressed[5]/chunked) version of the chain hosted on cheap object storage[6].

The existing internal developer tool immdb-server can be seen as a minimal prototype of the Genesis Sync Accelerator. Concretely, a user wanting to sync a node can download the historical chain from an untrusted source and expose it via immdb-server for consumption via their syncing node. Naturally, this isn't already ideal, such as the peak disk usage being twice the total chain size.

[5] Also see draft CIP https://github.com/cardano-foundation/CIPs/pull/993.
[6] For example Cloudflare R2, offering free egress.

As part of this proposal, a more optimized approach that can download *chunks* of the historical chain is to be designed and implemented. In particular, this allows to efficiently support the common use case of syncing nodes that already have some of the historical chain (which usually happens when a user didn't start their node for a prolonged period of time). Also, the disk usage of such a tool will only be a small constant.

Dependencies

Likely discussion in the Consensus and Network Working Groups.
Potential synergies with Mithril, especially around hosting the historical chain.
Discussions with full wallets like Daedalus for integration.

Maintenance

Low for the source code, only minor work expected for new eras.

Of course, any party hosting the necessary files for the historical chain has continuous (albeit very small) expenses for the hosting provider (both serving the files and periodically updating them as the historical chain grows), as well as costs associated with monitoring the service; however, these costs are not directly part of this proposal.

Key Proposal Deliverables

- An implementation of the Genesis Sync Accelerator proxy.
- Documentation, deployment advice and tooling for hosting the files consumed by the Genesis Sync Accelerator.

Resourcing & Duration Estimates

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Genesis Sync Accelerator	2	6	$27,680	$332,160

Project Name: Canonical Block and Transaction Diffusion Codecs

Section 3: Problem Statements and Proposal Benefits

Problem Statement

The Consensus Team has recently identified two implementation details of the Cardano node that currently leak into its contract with peers: Epoch Boundary Blocks (EBBs) and ledger era tags. EBBs have confused and disrupted developers many times since the beginning of Cardano and continue to spoil some intuitive invariants—most notably that there is at most one block per slot. Ledger era tags have also caused some confusion, since their relation to major protocol versions is not explicitly obvious: some protocol version increments also increment the ledger era and some do not. Especially with the advent of non-reference node implementations, the evolution of the ledger era tags may become a burden for multi-node developer committees to manage and even more confusing as a result.

Proposal Benefit

This proposal would remove EBBs from the historical chain (despite each being half of a megabyte, EBBs do not at all influence subsequent ledger states) and remove ledger era tags from the CBOR codecs of the node, replacing them instead with the major component of the protocol version, which is clearly not merely an implementation detail. These simplifications will make it easier for non-reference node implementations to reach sufficient parity to interoperate with the Cardano node today as well as to interoperate in the future as the Cardano protocol continues to evolve with incremented protocol versions—EBBs and ledger tags are footguns that should be removed earlier rather than later.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

It supports the product roadmap

Does your proposal align to any of the categories listed below?*

Core

Committee Alignment

TSC

Section 4: Proposal Details

Proposal Name

Canonical Block and Transaction Diffusion Codecs

Proposal Description

The canonical representation of on-chain data should exclude EBBs and it should include major protocol versions instead of ledger era tags. The proposed work achieves that by both retroactively removing EBBs from the historical Cardano chain as well as replacing ledger tags within the CBOR encoding of Cardano's blocks, block headers, and transactions.

- The CIP in Pull Request 974 specifies how EBBs would be removed. It remains in draft status today merely for the orthogonal issue of how to permanently archive the EBBs for future reference and the sake of transparency.
- Altering the CBOR codecs to contain the major protocol version instead of the ledger era tag is trivial: both are small integers. However, the (reference) Cardano node had its reasons to use ledger eras in the first place, and replacing them with the protocol version will require alterations within the node itself to compensate for the resulting architectural inconveniences.
    - In particular, replacing the coarser ledger era tags with the finer protocol versions in transactions implies a non-trivial change to the user interface of the cardano-cli tooling. The expected plan is for transactions to carry a protocol version only optionally: the Ledger Team takes great care when evolving the transaction codecs (since transactions are ultimately are part of the blocks), and as a result users submitting transactions to the network would almost never need to explicitly state which protocol version / ledger era they intended for their transaction: the new codecs are usually fully backwards-compatible.
    - Replacing ledger era tags with major protocol versions in the envelope for blocks and block headers is less radical than the change for transactions, but would almost certainly require changing the (reference) Cardano node's schema for storing its chain on-disk.
- The Cardano node's on-disk schema for chains has never needed a backwards-incompatible change before, so this schema simplification would be the first schema migration. This proposal bundles EBB removal and ledger era tag replacement precisely because the on-disk schema could be additionally simplified in the absence of EBBs: once they're gone, it could actually rely on the expected invariant that each Ouroboros slot contains at most one block. The proposed work would therefore set a precedent for schema migrations within the Cardano node by orchestrating the first two such migrations.

Dependencies

- Both removal plans will involve a sequence of new handshake versions, which always need to be coordinated with the Consensus Team and Network Team.
- The on-disk schema tends to be stable, so plenty of tools scrutinize it today despite it explicitly being an internal detail rather than part of the node's versioned stable interface. These simplifications will ultimately benefit more than just the node architects, but they'd also need to be announced well in advance (via a CIP, eg) and potentially require advising some community members how to update their own tooling.
- The new codecs would eventually need to be implemented by all nodes, not just the reference node. The ultimate benefit of this proposal is to simplify the requirements of every node's design, so hopefully the node authors will agree the necessary changes are worthwhile.

Maintenance

- The finite series of handshake versions will need to be orchestrated amongst the releases of node(s) in order to execute the proposed work. This will be simplest to do if the proposed work is complete before a significant portion of nodes are using non-reference implementations.

Key Proposal Deliverables

1. A series of peer-to-peer/peer-to-client handshake version refinements that ultimately removes implementation details from the node's codecs for diffusing (pieces of) its chain.
    a. See the CIP in Pull Request 974 for details about EBB removal.
    b. For ledger era tags, the proposed work will include drafting a CIP that specifies the changes—including review from architects of non-reference nodes—and then implementing those changes in the (reference) Cardano node.
2. A schema migration for the node's internal on-disk storage schema to simplify it once those implementation details have been hidden.
3. A general plan recommending how to approach future migrations of the node's on-disk schema (eg lessons learned and pain points).

Resourcing & Duration Estimates

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Canonical Block and Transaction Diffusion Codecs	3	6	$27,680	$498,240

Disclaimer: although eliciting attention and approval of node architects/maintainers and any requisite deprecation cycles might spread out the ultimate arrival of these deliverables on mainnet

Project Name: Hoarding Node

Section 3: Problem Statements and Proposal Benefits

Problem Statement

Blocks and transactions that do not eventually end up on the historical chain are (by design) not permanently stored by the Cardano node. However, for monitoring and investigation purposes, it is often desirable to be able to access such data in an easy manner. Concrete examples include:

- Bugs regarding disagreement on block/tx validity between different nodes (example) or crashes due to certain transactions (example). In the future, such bugs might be caused due to subtle differences between different node implementations.
- Automatic mempool monitoring for undesirable transactions, such as those causing the June 2024 DoS attack.
- Identifying, evidencing, understanding and effectively responding to adversarial behavior, such as induced rollbacks, diffusion of invalid blocks and transactions or block equivocation attacks.
- Analytical use cases, such as (apparently non-adversarial) behavior of poorly configured/underresourced nodes (potentially indicating optimization opportunities), as well as timing of blocks (late/early) and the frequency of slot and height battles.

For this purpose, we propose to design and implement a “hoarding node” that connects to various nodes in the Cardano network and stores all (up to a configurable limit) blocks and transactions, even if they are invalid or orphaned.

Proposal Benefit

A hoarding node improves the ability to effectively respond to unforeseen situations in the Cardano network, and gives further empirical insights into its complex dynamics.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Doesn’t directly align

Does your proposal align to any of the categories listed below?*

TBD

Committee Alignment

TSC

Section 4: Proposal Details

Proposal Name

Hoarding Node

Proposal Description

The goal of the proposal is to implement a “hoarding” node dedicated to storing all (up to a configurable limit) observed headers, blocks and transactions. To this end, the hoarding node will act just like a normal relay node in the Cardano P2P network, establishing both outbound (for headers/blocks) and inbound connections (for transactions), just like other normal relay nodes hosted by parties interested in following the chain and allowing inbound connections.

By default, only headers/blocks/transactions are stored that do not end up on the historical chain, minimizing the storage cost in the optimistic case. Adversarial behavior (such as block equivocation) causes the amount of storage to be trivially unbounded, so appropriate heuristics are to be implemented (such as only storing at most a bounded amount of blocks per election opportunity).

A simple initial prototype will not do any validation whatsoever, and hence rely on a separate full node to maintain its selection for diffusion to downstream peers. Further refinements include header and even block validation, recording their validity for easy retrieval. This also opens the opportunity to automatically detect and flag unusual (e.g. expensive-to-validate) blocks/transactions, enriching the monitoring capabilities of a hoarding node.

Any data collected by the hoarding node is exposed in an easy-to-consume API (e.g. metrics for observability frameworks, and an HTTP API for fine-grained further analysis).

We stress that the load of such a hoarding node on honest nodes is equal to that of any node following the chain. Still, it is possible to allow honest stake pools to opt-out of being connected to by a particular deployment of a hoarding node.

Finally, it is conceptually unavoidable that adversarial behavior can only be recorded on a best-effort basis by the hoarding node, e.g. because adversaries can use heuristics to serve

different blocks/transactions to hoarding nodes compared to honest stake pool relays. However, all adversarial behavior that is relayed by honest nodes will also be recorded by a hoarding node.

Discussion of prior art and related services

The basic idea of a service as outlined in this proposal has been around for a long time, see e.g. here. A simple version (only for headers/blocks, not transactions) has been prototyped as part of cardano-slurp.

https://pooltool.io/ has overlap in the analytical aspects with the hoarding node. Fundamentally, these tools are complementary, as pooltool relies on opt-in information sent by stake pools, whereas a monitoring node does not.

Dependencies

Discussions in the Network Working Group on the design

Maintenance

Low, needs to be updated for breaking changes in the network protocol (i.e. hard forks).

Key Proposal Deliverables

- A design for a hoarding node, describing in particular how the extra load on honest nodes is kept minimal.
- An initial implementation using the existing Haskell network stack, together with an easy-to-consume API.

Resourcing & Duration Estimates

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Hoarding Node	3	6	$27,680	$498,240

Project Name: Block Cost Investigation

Section 3: Problem Statements and Proposal Benefits

Problem Statement

The parameter values of Ouroboros Praos for its specific instantiation on Cardano mainnet were tuned many years ago. One such parameter is Δ, the worst-case delay for a fresh honest block to be reliably adopted by the producer of the next honest block. The transactions and blocks have become much more sophisticated since those tuning decisions, most notably with Plutus scripts in the Alonzo era and reference inputs in the Babbage era. It is overdue to investigate whether the adversary might be able to mint a worst-case block that is so resource intensive for honest nodes today to validate that the actual worst-case message delay is much greater than the values used so far for Δ.

Recent relevant examples include the June 2024 attack and a performance bug detected shortly after the Chang hard fork to the Conway era (the potential DoS vector mentioned in the 10.1.4 release notes). Could a more sophisticated adversary have done much more harm?

Proposal Benefit

This investigation would derisk the potential modern inaccuracy of the Δ values. The development of the investigation itself will also yield some methodologies and/or benchmarks to help catch future regressions to the worst-case block cost before they reach mainnet.

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Leios
Scaling the L1 Engine

Does your proposal align to any of the categories listed below?*

Core

Committee Alignment

TSC

Section 4: Proposal Details

Proposal Name

Investigate Worst-Case Block Cost and Develop Regression Test Methodology

Proposal Description

There are four pillars of this work.

- Deriving an empirical cost model from all blocks on mainnet.
    - Define a set of *features* such as number of transactions, number of TxIn, byte size of reference scripts, etc.
    - Use some *regression* and *model selection* techniques to choose and fit a formula for the time and allocation costs in terms of those features.
    - Fitting this model will be most useful if it is interpretable–indicating where to look for performance bugs. It's unclear what value an uninterpretable model (eg a traditional deep learning neural network) would have.
- Deriving an expected cost model from first principles and the ledger rules and consensus protocol themselves.
    - This should not be comprehensive or exact but instead focus on the known characteristics and usual suspects: large UTxO map, large delegation map, expensive cryptographic operations, incremental calculations, UTxO HD disk bandwidth, and so on.
    - Ideally, the empirical and theoretical models would agree, such that any observed disagreements (eg linear instead of quadratic) would indicate bugs or confusion.
- Prescribing a general method for revising and/or extending those models as the ledger rules and consensus protocol evolve—primarily: lessons learned and pain points during the investigation that seem likely to apply in the future.
- Designing and implementing automatic anomaly detectors for mainnet as well as regression suites for use in Continuous Integration (CI), both via recent mainnet blocks and also synthetic block content.
    - The synthesizers should not be tuned to reflect mainnet, but rather to reflect extreme data.
    - A search via gradient ascent, genetic programming, etc might be called for, as opposed to the generators without any feedback used in most property testing.
    - Finally, it's worth explicitly noting that synthetic block context may potentially find performance bugs in the implementation of a new ledger era's features before mainnet forks into that era (which might have instead entirely prevented the DoS vector eliminated by the 10.1.4 release).

The results of this work may also be useful for informing or judging incentivization schemes such as the transaction fee formula and transaction/block measure limits. However, it is crucial to understand that some adversaries can afford otherwise-prohibitive fees if the success probability is high enough. Even more crucially, if the attack gives the adversary control of the chain, they may be able to finish it by rolling back the expensive blocks, thereby ultimately avoiding all of the fees—as long as they could put up sufficient fees during the attack.

New protocol features could also leverage these cost models. The Leios Innovation Team is currently simulating hypothetical traffic amongst the entire network, and their simulations must make some assumptions about the costs of validating transactions, blocks, etc. The empirical model could also easily be separately fitted to the time necessary to construct the ledger state that results from applying a transaction/block under the assumption that it is valid. The theoretical model could feasibly also branch on full-validation versus mere result-construction, but that'd be more complex to specify correctly—on the other hand, agreement with the empirical model would provide some evidence.

Dependencies

- This work would be considered part of the "quantitative time agreements (QTA)" work that has already been proposed within the existing Cardano development teams but not yet initiated.
- The IOG Ledger Team and IOG Consensus Team would provide invaluable input throughout each phase. In particular, both teams will also need to contribute to the design of the regression suites for CI.
- Well-Typed and some GHC experts would likely significantly accelerate some sub-investigations that are very likely to arise.
    - Well-Typed has had teams working on tools for analyzing the performance of the GHC garbage collector for multiple years.
    - It is notoriously difficult to predict garbage collection overheads with any granularity. Well-Typed and the broader GHC community has some of the world's experts.

Maintenance

New ledger and consensus features would need to be integrated into the theoretical model, anomaly detectors, and regression suites. Eventually, they could be integrated into the empirical model as well, once they have a significant presence on mainnet.

Key Proposal Deliverables

1. Report explaining the empirical cost model and its derivation from mainnet blocks.
2. Report explaining the theoretical cost model and its derivation from the specifications.

3. Report explaining a general method for revising and/or extending those models as the ledger rules and consensus protocol evolve.
4. Specification and implementation of automatic anomaly detectors for mainnet.
5. Specification and implementation of regression suites for use in Continuous Integration, both via empirical mainnet blocks and also synthetic block content.

Resourcing & Duration Estimates

Project name	F	Effort in Months	Price per month per FTE	Project price
Block Cost Investigation	3	8	$27,680	$664,320

Project Name: Cardano-node-emulator: Maintenance proposal

I. Introduction

Context

cardano-node-emulator (CNE) is a tool designed to simulate certain behaviours of the Cardano node without relying on an actually deployed node. It is principally used for the validation and testing of Plutus based smart contracts. CNE has spawned from the old mono-repo plutus-apps which was put into maintenance mode two years ago, and is now archived since December 2024. This repository was unmaintainable due to its monolithic structure and the loss of relevance of some of its components. However, among those components, a few remained relevant, and have been placed within CNE. In particular, CNE contains *plutus-script-utils* which provides the notion of typed validators, a notion that used to be almost universally used when defining plutus script, and that has slowly fallen into disuse.

CNE is currently used by various companies for testing purposes, and specifically by Modus Create which makes use of the tool in their own open source testing tool, cooked-validators. However, despite CNE being the subpart of *plutus-apps* that was supposed to be kept under development, maintenance of CNE has been reduced to the bare minimum, if not stopped altogether. As the Cardano ecosystem is evolving rapidly, with the recent arrival of the Chang hard fork and various API changes such as the ones described in CIP69, an increased effort of development maintenance is required to stop CNE from being abandoned.

Proposal

Modus proposes to take over principal maintenance of cardano-node-emulator. This effort will be twofold: first, we propose to close the gap between the current version of cardano-api and the current version of CNE. Then, we propose to conduct continuous maintenance activities to keep CNE up to date and usable with respect to the evolution of Cardano.

Motivation

While the context described above might indicate that CNE should be archived in favour of newer libraries and tools to build and test smart contracts, there are two main reasons why we believe that it should, instead, be updated and maintained.

The first reason revolves around the emulator itself. Running a node is costly, complicated and does not allow for efficiently sending a significant amount of transactions in a short period of time, which is essential for testing the correctness of smart contracts. Having a proper emulator

which gets rid of this central need is a convenient and efficient substitute, which was the original intent of CNE.

The second reason revolves around the utilities contained within CNE, such as the infrastructure around typed validators. The use of typed validators has slowly decreased due to the following reasons:

- Type safety often reduces flexibility, and relying on BuiltinData allowed developers to regain this flexibility, while losing in clarity and readability of their codebase.
- Developers of smart contracts realized that deserializing the full context, redeemer, and datum of a script was responsible for a non-negligible portion of the resource consumption of the script, thus increasing the transaction fee.
- The old typed validators no longer comply with CIP69 requirements, in particular with the ability of a script to be used with various purposes, and thus have a variable redeemer type.
- Developers moved away from Plutus itself, using Plutarch or other languages compiling to plutus-core.

While we do agree that CNE as it stands does suffer from these limitations and is seriously outdated, the core idea between the utilities it provides still remains. Type safety in a Haskell ecosystem is key to correctness, and we believe that we can recapture that original idea and package it in a way that suits the current standard and needs of Cardano developers.

Scope

This proposal pertains to the code and ancillary documents hosted in the following repository: https://github.com/IntersectMBO/cardano-node-emulator. Note that quickcheck-contractmodel is out of scope as it is only used by CNE for testing contracts. As we do not intend to maintain it, and this library is only used for testing purposes, we propose to get rid of it altogether.

Staffing

We propose two maintainers, who will share the work as fits their particular skills and as other commitments dictate.

Mathieu Montin is responsible for the smart contract auditing activity within Modus create, and was the leader of the former high assurance software team within Tweag. He has significant experience with Plutus and Cardano. He is the main developer and maintainer of cooked-validators, which itself uses CNE to assist in auditing Plutus contracts.

Sjoerd Visscher is an engineer and prominent Haskell contributor. He has significant experience with the Plutus community, and is the biggest existing contributor to cardano-node-emulator.

II. Initial update of cardano-node-emulator

CNE is currently outdated compared to the current version of cardano-api and associated libraries. We propose to close this initial gap through the following tasks:

- Updating the direct dependencies to the relevant APIs, and match their version to the ones deployed on Cardano Haskell Packages (CHAP).
- Updating the inner state on which the emulator relies, reaching an emulated ledger state that is closer (if not identical) to the actual ledger state from cardano.
  In practice, this will start by updating and eventually merging the following pull request (if it can be properly updated): https://github.com/IntersectMBO/cardano-node-emulator/pull/19 and also remove the duplicated state to solely rely on Cardano state. This will make it much easier in the future to follow possible changes into the ledger state. This step is crucial as Conway introduced many additional elements into this state.
- Rethinking and updating the typed validators and various wrappers in plutus-script-utils to match the new kinds of scripts following Conway and CIP69. In particular, the wrappers Validator and MintingPolicy - which limit scripts to have a single purpose - might be outdated and we should rethink their design, as scripts are now allowed to be multi-purpose.
- Release CNE on CHAP so that the available version there matches the current standards. This will attract new users and improve efficiency of builds for external projects depending on this library. Depending on the various dependencies, this release can either be very quick or rather cumbersome ("cabal hell").

For the 4 tasks depicted above, we propose a total duration of 20 days of work.

Modus has already developed a draft pull request to act as a proof of concept for this initial update.

III. Regular maintenance

Maintenance responsibilities

After the initial phase of update, we propose to take over principal maintenance of the above CNE repository. In particular, this would extend to the following activities:

1. **Maintaining the backlog and roadmap**
   a. Develop and publicise a medium-term roadmap for the development of the cardano-node emulator, based upon the results of planning, community discussion and tracked issues.
   b. Based upon this roadmap, we will maintain a short-term backlog of work items.

2. **Reviewing and Merging PRs**
    a. Monitor the active PR lists and shepherd them to completion.
    b. Assist external contributors in making contributions.
3. **Stakeholder and Community Engagement**
    a. Identify and maintain the list of key stakeholders for the project
    b. Create channels for contributors to engage with developing the backlog and priorities for the project.
    c. Identify work items particularly suitable for work on by newcomers and external contributors.
    d. Regularly inform stakeholders about the state of the project.
4. **Establishing clear project governance**
    a. Establish project governance structures according to the Intersect governance framework (https://intersect.gitbook.io/open-source-committee/policies/governance)
    b. Establish project documentation as per the Intersect documentation framework (https://intersect.gitbook.io/open-source-committee/policies/documentation)
5. **Perform regular maintenance**
    a. Work on the backlog items as identified in (1)
    b. Keep the project up-to-date with dependencies - in particular, with the current iteration and era of cardano-node.
    c. Maintain a CHANGELOG of all major changes to the codebase.

Detailed projects maintenance plan

CNE repository contains various projects, each of which does not require the same level of maintenance work. Here is a brief overview of those specific needs for each of them:

- **freer-extra**: this is a library that is unrelated to CNE per se, but is used by it and was not present elsewhere, so was created for the occasion. Should not require any maintenance because it is unconnected to other Cardano libraries.
- **cardano-node-emulator-socket**: this is an interface library to allow for using the emulator instead of the real node in practice. It misses a lot of features and is overall in a stand-by state. We propose only low-maintenance for this project. A decision should be made in the future whether it should be maintained or be stopped, depending on the number of users.
- **plutus-script-utils**: active maintenance mode. This provides various utilities which we propose to keep up to date and enlarge as Cardano changes.
- **plutus-ledger**: this contains legacy code mostly, with helpers to bridge Plutus and Cardano. We propose a low maintenance mode for this library. Ultimately, we should decide whether to drop it completely, or how to spread useful parts around, for example by putting them directly into cardano-node-emulator.
- **cardano-node-emulator**: this is the main component of the repository, which consists of the emulator itself. It should of course be put into active maintenance.

Stakeholders

We have already identified the following individuals and/or groups as key stakeholders to the project:

- **Quvik** are the original developers of quickcheck-contractmodel. They are currently not maintaining the library but have a number of projects that still have quickcheck-contractmodel as a dependency.
- The **Plutus-HA** team within IOE rely on the cardano-node-emulator.
- **Tweag** relies on the cardano-node-emulator for their cooked-validators project.
- The **cardano-node** team is responsible for upstream developments in the node which must be reflected in the node emulator.
- Researchers in Philip Wadler's group are using the cardano-node-emulator.

Communication Channels

As part of the maintainer role, the team will engage and communicate with stakeholders and the community in order to guide priorities and work. While the details of the channels needed may evolve over time, we propose the following initial communication channels:

- The project plan and backlog will be principally tracked via issues (https://github.com/IntersectMBO/cardano-node-emulator/issues) on the cardano-node-emulator repo. Should additional structure be needed, we will organise this using projects in the same repository.
- We will establish and monitor a discord channel on the Intersect discord server which anyone interested in the project may join.
- We propose a quarterly stakeholder meeting in which we discuss work done, the project plan and seek input from any interested parties. This will be advertised ahead of time in the above discord channel with direct invitation to stakeholders. The frequency of this meeting may be increased as needs demand, but should not be any less regular than quarterly.

Price

Project name	FTE	Effort in Months	Price per month per FTE	Project price
Cardano-node-emulator proposal	2	12	$27,680	$166,080

- Initial Phase: 20 engineer days dedicated to project setup and initial development.
- Ongoing Support: Continuous development, updates, and troubleshooting over the next 12 months.

This comprehensive pricing reflects our commitment to delivering exceptional value and sustained collaboration.

Project Name: History Expiry

Section 3: Problem Statements and Proposal Benefits

Problem Statement

Ouroboros - the snake perpetually devouring its own tail. But that tail is growing ever larger, and it's taking longer and longer for the snake to catch up to its own head - not to mention the amount of tail that must be devoured along the way. It's enough to give a snake indigestion!

The current cardano node can be seen as fairly unique in its approach to upgrades. While other networks need to effectively stop and restart in order to deal with upgrades to the core logic, the hard fork combinator and polymorphic ledger have allowed Cardano to seamlessly progress from Byron through to the current Conway era, changing consensus protocols along the way. The current node may just as easily act as a Byron era node.

However, this benefit has a downside - the Cardano chain now stretches back over 7 years, and this full history must be processed by any node joining the network. Each node is equally expected to be able to serve the historical chain should it be requested. This has a number of costs:

1. Time to sync a node from scratch is now over a day and growing.
2. Significant storage requirements for each node, many tens of gigabytes on disk.
3. High network traffic requirements on syncing nodes and the upstream peers serving them.

Requiring honest nodes to be able to serve the historic chain is also a potential risk for new node implementations, most of which do not intend to implement the logic of long-past eras.

Once Leios is implemented, increased transaction volume will greatly exacerbate this problem.

Proposal Benefit

With the advent of tools such as Mithril to allow nodes to bootstrap from snapshots, we have the possibility to now allow history to be truncated. This would allow honest cardano nodes to compress and forget all history past a certain point. There would be a number of benefits:

- Significantly faster syncing speed (even Mithril snapshots would be smaller, since there would be no need to distribute the entire historical chain)
- Lower hardware requirements for node storage.
- Lower network costs to synchronize historic state.

- Alternate node implementations would not risk being “technically malicious” due to refusing to serve historical data or else serving historical data that they have not personally validated!

Does this proposal align to the Product Roadmap and Roadmap Goals?*

Multiple Node Implementations
Scaling the L1 engine

Does your proposal align to any of the categories listed below?*

Core

Committee Alignment

TSC

Supplementary Endorsement

- The discussion on https://github.com/cardano-foundation/CIPs/pull/974 endorses this plan. Note that this plan would probably render that CIP redundant.
- Amaru is not planning to support old eras

Section 4: Proposal Details

Proposal Name

History Expiry

Proposal Description

We propose the following work:

1. A reformulation of honest behaviour to support nodes which do not offer the historical chain.
2. Updates to chain sync and block fetch protocols to explicitly support communication about “pre-historic” chains and their availability.
3. Updates to the node to support explicitly starting from a snapshot with no historical chain.
4. Logic in the Cardano node to support moving the historical genesis anchor (e.g. further truncating the chain).

In addition, once regular nodes are running without historical state, we envision the need for specific archive nodes serving the historical chain. Thus we also plan to deliver:

1. A design for the incentives to support running archival nodes.
2. A bare bones archival node which will serve the historical chain in a format suitable for downstream tools.

Dependencies

- Canonical ledger state would be helpful for a few things:
    - Makes the Mithril bootstrap sequence much safer
    - Allows potential use in communicating ledger states rather than blocks
- Canonical Block and Transaction Diffusion Codecs would need to modify the archival node as well, but that would reduce the amount of reference node implementation details in the messages the archival node serves to peers that want to sync the pre-historical chain.
- Both this proposal and Canonical Block and Transaction Diffusion Codecs will eventually require a schema for publicly archiving data: at least EBBs and the source code/binaries/etc of older versions of the node that can validate the pre-historic chain.

Maintenance

The main thrust of this work will actually be to reduce the maintenance burden on the Cardano node. Large chunks of legacy code may be removed, reducing the size of the codebase to maintain. Old versions of the node may be maintained in order to process and verify the historical chain.

Some maintenance may be needed for the archival node, but we envision that this should be designed in such a way as to have minimal logic that will need updating for future eras.

Key Proposal Deliverables

- CPS and CIP setting out the full scope of work
- Update network protocols with support for pre-historic chains.
- Updated Cardano node with support for pre-historic chains and starting from a snapshot.
- Incentive design for archival nodes.
- Simple archival node.

Resourcing & Duration Estimates

Project name	F	Effort in Months	Price per month per FTE	Project price
History Expiry	4	9	$27,680	$996,480

Cost breakdown:

Based on your preferred contract type and cost estimate, a cost breakdown in ADA and in USD.*

No.	Project name	FTE	Effort in Months	Project price in USD	Project price in ADA ($0.6601 = ₳ 1)
1	Peras	7	10	$1,937,600	₳ 2,935,312.83
2	Canonical Ledger State	3	4	$332,160	₳ 503,196.49
3	Black box Ledger Conformance Testing	3	4	$332,160	₳ 503,196.49
4	Conformance Testing of Consensus	4	9	$996,480	₳ 1,509,589.46
5	Plutus Script Re-Executor	4	9	$996,480	₳ 1,509,589.46
6	Genesis Sync Accelerator	2	6	$332,160	₳ 503,196.49
7	Canonical Block and Transaction Diffusion Codecs	3	6	$498,240	₳ 754,794.73
8	Hoarding Node	3	6	$498,240	₳ 754,794.73
9	Block Cost Investigation	1	8	$221,440	₳ 335,464.32
10	Cardano-node-emulator proposal	2	3	$166,080	₳ 251,598.24
11	History Expiry	4	9	$996,480	₳ 1,509,589.46
	Total value of Tweag proposals			**$7,307,520**	**₳ 11,070,322.68**

Contact

Thank you for taking the time to review our project proposals. We appreciate the opportunity to further cooperation with the Cardano Community

If you have any questions or require additional information, please do not hesitate to contact us.
Kris Kowalsky (kristijan.kowalsky@moduscreate.com) or Nicholas Clarke (nicholas.clarke@moduscreate.com)
We welcome the opportunity to discuss this projects further.

Thank you again for your consideration.

Kristijan Kowalsky
Client Partner

Nicholas Clarke
Director of Engineering